

AB-40

APPLICATION BRIEF

32-Bit Math Routines for the 8051

RICK SCHUE REGIONAL APPLICATIONS SPECIALIST INDIANAPOLIS, INDIANA

October 1992

Order Number: 270530-002

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation P.O. Box 7641 Mt. Prospect, IL 60056-7641 or call 1-800-879-4683

COPYRIGHT @ INTEL CORPORATION, 1996

32-BIT MATH ROUTINES CONTENTS PAGE FOR THE 8051 APPLICATION 1 CODE SOURCE LISTINGS 2

AB-40



Here are some easy to use 16- and 32-bit math routines that take the pain out of calculations such as PID loops, A/D calibration, linearization calculations and anything else that requires 32-bit accuracy.

The package is written to interface with PL/M-51. Parameters are passed as 16-bit words to the routines, which perform operations on a 32-bit "accumulator" resident in memory. The following functions are performed:

Load_16 (word_param)

Loads a 16-bit -RD into the low half of the 32-bit "accumulator", zeros upper 16 bits of accumulator.

Load_32 (word_hi,word_lo)

Loads word_hi into upper 16 bits of accumulator, word lo_into Lower 16 bits.

Low_16

Returns the lower 16 bits of the accumulator, bits 0 through 15.

Mid_16

Returns the middle 16 bits of the accumulator, bits 8 through 23.

High_16

Returns the upper 16 bits of the accumulator, bits 16 through 31.

Mul_16 (word_param)

Multiplies the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Div_16 (word_param)

Divides the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Add_16 (word_param)

Adds the 16-bit word supplied to the 32-bit accumulator.

Sub_16 (word_param)

Similar to Add_16 but for subtraction.

Add_32 (word_hi,word_lo)

Forms a 32-bit value for word_hi and word_lo and adds it to the accumulator.

Sub_32 (word_hi,word_lo)

Similar to Add_32 but for subtraction.

APPLICATION

Typical applications have 16-bit "input" values and produce 16-bit "output" values, but require 32-bit values for intermediate results. An example would be reading a 12-bit A/D, performing some gain and offset calculation on the raw A/D data to produce a calibrated 16 bit result. Doing this is a simple task with this math package.

```
CALL Load_16(AD_value);
CALL Add_16 (offset_value);
CALL Mul_16 (gain_factor);
/* gain is in units of 1/256 */
result = Mid_16;
```

In this example the accumulator was loaded with the raw A/D value and then the offset was applied. The gain_factor was "pre-multiplied" by 256 (8 bits), giving it a granularity of 1/256. The result was extracted from the "middle" 16 bits of the accumulator (bits 8 to 23) to account for the scaling factor of 256 introduced in the multiply step.

The package requires about 384 bytes of ROM and 30 bytes of RAM. Individual routines can be deleted to conserve RAM if they are not used.

1



CODE SOURCE LISTINGS

```
CODE SOURCE LISTINGS
 NAME Math_32_Module
                       Load 16, ?Load 16?byte
Load 32, ?Load 32?byte
Mul 16, ?Mul 16?byte
Div 16, ?Duv 16?byte
Add 16, ?Add 16?byte
Sub 16, ?Sub 16?byte
Add 32, ?Add 32?byte
Sub 32, ?Sub 32?byte
Low 16, Mid 16, High 16
  PUBLIC
 PUBLIC PUBLIC
  PUBLIC
  PUBLIC
  PUBLIC
  PUBLIC
  PUBLIC
 PUBLIC
 Math_32_Data
Math_32_Code
                                             SEGMENT
                                                                             DATA
                                          SEGMENT
                                                                             CODE
 RSEG Math 32 Data
?Load 16?byte: DS 2
?Load 32?byte: DS 4
?Mul 16?byte: DS 2

    ?Mul 16?byte:
    DS 2

    ?Div_16?byte:
    DS 2

    ?Add 16?byte:
    DS 2

    ?Sub_16?byte:
    DS 2

    ?Add 32?byte:
    DS 4

    ?Sub_32?byte:
    DS 4

    OP 0:
    DS 1

    OP 2:
    DS 1

    OP 2:
    DS 1

    OP 3:
    DS 1

 OP_3:
TMP_0:
                                              DS 1
                                             DS 1
 TMP_1:
 TMP 2:
                                             DS 1
 TMP_3:
                                              DS 1
 RSEG Math_32_Code
                                                                                                                                                                                                               270530-1
```

intel_®

```
Load 16:
;Load the lower 16 bits of the OP registers with the value supplied
     MOV OP 3,#0

MOV OP 2,#0

MOV OP 1,?Load 16?byte

MOV OP 0,?Load 16?byte + 1
Load_32:
     ;Load all the OP registers with the value supplied
     MOV OP 3,?Load 32?byte MOV OP 2,?Load 32?byte + 1
MOV OP 1,?Load 32?byte + 2
MOV OP 0,?Load 32?byte + 3
     RET
Low_16:
     ;Return the lower 16 bits of the OP registers
MOV R6,OP 1
MOV R7,OP 0
     RET
Mid_16:
     Return the middle 16 bits of the OP registers
             R6,OP_2
R7,OP_1
     MOV
     MOV
     RET
High_16:
     ;Return the high 16 bits of the OP registers MOV R6,OP_3 MOV R7,OP_2
     RET
Add_16:
     ;Add the 16 bits supplied by the caller to the OP registers
     ÇLR
     MOV
              A,OP 0
     ADDC A,?Add_16?byte + 1 ;low byte first
             OP_0,A
A,OP_1
     MOV
     MOV
     ADDC
             A,?Add_16?byte
                                        ;high byte + carry
     MOV
              OP_1,A
             A,OP_2
     MOV
     ADDC
             A,#0
                                         ;propagate carry only
            OP_2,A
A,OP_3
A,#0
     MOV
     MOV
     ADDC
                                         ;propagate carry only
     MOV
             OP_3,A
     RET
                                                                                                          270530-2
```



```
Add_32:
    ;Add the 32 bits supplied by the caller to the OP registers
    CLR
           A,OP 0
    MOV
          A,?Add_32?byte + 3 ;lowest byte first
    ADDC
    MOV
           OP_0,A
    MOV
           A,OP_1
          A,?Add_32?byte + 2
                                ;mid-lowest byte + carry
    ADDC
    MOV
           OP_1,A
    MOV
           A,OP 2
          A,?Add_32?byte + 1
OP 2,A
                                ;mid-highest byte + carry
    ADDC
    MOV
    MOV A,OP 3
ADDC A,?Add_32?byte
                                 ;highest byte + carry
    MOV
           OP_3,A
Sub 16:
    ;Subtract the 16 bits supplied by the caller from the OP registers
    CLR
           A,OP_0
    MOV
    SUBB A,?Sub_16?byte + 1
                                 ;low byte first
    MOV
           OP O,A
    MOV
           A,OP_1
    SUBB A,?Sub_16?byte
                                  ;high byte + carry
    MOV OP 1,A
MOV A,OP 2
SUBB A,#O
MOV OP 2,A
MOV A,OP 3
SUBB A,#O
                                  ;propagate carry only
                                  ;propagate carry only
    MOV
           OP_3,A
Sub_32:
    ;Subtract the 32 bits supplied by the caller from the OP registers
    CLR
    MOV
           A,OP_0
    SUBB
          A,?Sub_32?byte + 3
                                 ;lowest byte first
           OP_O,A
A,OP_1
    MOV
    MOV
           A,?Sub_32?byte + 2
    SUBB
                                 ;mid-lowest byte + carry
           OP_1,A
A,OP_2
    MOV
    MOV
    SUBB
           A,?Sub 32?byte + 1
                                  ;mid-highest byte + carry
           OP_2,A
A,OP_3
    MOV
    MOV
    SUBB
           λ,?Sub 32?byce
                                  ;highest byte + carry
    MOV
           OP_3,A
    RET
                                                                                        270530-3
```

intel_®

```
Mul 16:
    ;Multiply the 32 bit OP with the 16 value supplied MOV TMP_3,#0 ;clear out upper 16 bits MOV TMP_2,#0
    ;Generate the lowest byte of the result
          B,OP 0
    MOV
           A,?Mul_16?byte+1
    MOV
    MUL
           AB
    MOV
           TMP 0,A
                            ;low-order result
           TMP 1,B
                           ;high_order result
    MOV
    ; Now generate the next higher order byte
    MOV
          B,OP 1
    VOM
           A,?Mul 16?byte+1
    MUL
           AB
    ADD
           A,TMP_1
                           ;low-order result
                           ; save
    VOM
           TMP_1,A
    MOV
           A,B
                           ; get high-order result
          A,TMP_2
    ADDC
                            ; include carry from previous operation
           TMP 2,A
Mul_loopl
TMP_3
    MOV
                           ; save
    JNC
    INC
                           ; propagate carry into TMP 3
Mul_loop1:
    MOV
           B.OP 0
    MOV
           A,?Mul_16?byte
           AB
    MUL
    ADD
           A,TMP 1
                            ; low-order result
           TMP_1,A
    MOV
                           ; save
    MOV
           A,B
                           ; get high-order result
    ADDC
          A,TMP 2
                           ; include carry from previous operation
           TMP 2,A
Mul loop2
TMP 3
    MOV
    JNC
                          ; propagate carry into TMP 3
    INC
Mul_loop2:
    ; Now start working on the 3rd byte
    MOV
          B,OP_2
    MOV
           A,?Mul_16?byte+1
    MUL
           AB
           A,TMP_2
TMP_2,A
A,B
    ADO
                           ;low-order result
    MOV
                          ; save
    MOV
                           ; get high-order result
           A,TMP_3
TMP_3,A
    ADDC
                           ; include carry from previous operation
    MOV
                           ; save
    ; Now the other half
    MOV
          B,OP_1
    MOV
           A,?Mul_16?byte
    MUL
           AB
           A,TMP_2
    ADD
                           ;low-order result
           TMP 2,A
A,B
    MOV
                           ; save
    MOV
                           ; get high-order result
          A,TMP 3
    ADDC
                           ; include carry from previous operation
           TMP 3,A
    MOV
                           ; save
    ; Now finish off the highest order byte
    MOV B, OP 3
           A,?Mul_16?byte+1
                                                                                       270530-4
```



```
MUL AB
ADD A, TMP 3 ; low-order result
MOV TMP 3,A ; save
; Forget about the high-order result, this is only 32 bit math!
MOV B,OP 2
MOV A,?Mul 16?byte
MUL AB
ADD A,TMP 3 ; low-order result
MOV TMP 3,A ; save
; Now we are all done, move the TMP values back into OP
MOV OP 0,TMP 0
MOV OP 1,TMP 1
MOV OP 2,TMP 2
MOV OP 3,TMP 3
RET

270530-5
```

intel_®

```
This divides the 32 bit OP register by the value supplied MOV R7,#0
Div_16:
           R6,#0
                              ; zero out partial remainder
    MOV
           TMP 0,#0
TMP 1,#0
TMP 2,#0
    YOV.
    MOV
    MOV
           TMP_3,#0
Rl,?Div_16?byte
    MOV
    MOV
                              ;load divisor
    MOV
           R0,?Div_l6?byte+l
    MOV
           R5,#32
                               ;loop count
    ;This begins the loop
Div_loop:
    CALL
           Shift_D
                           ;shift the dividend and return MSB in C
    VOM
           A,R6
                           ;shift carry into LSB of partial remainder
    RLC
    MOV
           R6,A
    MOV
           A,R7
    RLC
           Α
           R7,A
    MOV
    ;now test to see if R7:R6 >= R1:R0
                           ;Carry out of R7 shift means R7:R6 > R1:R0
    JC
          Can_sub
    CLR
    MOV
           A,R7
                           ;subtract Rl from R7 to see if R1 < R7
         A,R1
    SUBB
                           ; A = R7 - R1, carry set if R7 < R1
    JC
           Cant_sub
    ;at this point R7>Rl or R7=Rl
    JNZ
          Can_sub
                           ;jump if R7>Rl
    ;if R7 = R1, test for R6>=R0
    CLR C
    MOV A,R6
SUBB A,R0
                           ; A = R6 - R0, carry set if R6 < R0
    JC
           Cant_sub
Can_sub:
    ;subtract the divisor from the partial remainder CLR C
    MOV
           A,R6
                        ; A = R6 - R0
    SUBB
          A,RO
    MOV
           R6,A
    MOV
           A,R7
    SUBB A,R1
                        ; A = R7 - R1 - Borrow
    MOV
          R7,A
    SETB C
                         ; shift a l into the quotient
    JMP
          Quot
Cant_sub:
    ;shift a 0 into the quotient
    CLR C
Quot:
    ;shift the carry bit into the quotient CALL Shift_Q
    ; Test for competion
    DJNZ R5,Div_loop; Now we are all done, move the TMP values back into OP
    MOV OP 0, TMP 0
    MOV
           OP 1,TMP 1
                                                                                      270530-6
```



```
OP_2,TMP_2
OP_3,TMP_3
    MOV
    MOV
    RET
    ;shift the dividend one bit to the left and return the MSB in CCLR C
Shift_D:
    MOV
             A,OP_0
     RLC
            OP_0,A
A,OP_1
    MOV
     MOV
     RLC
             OP 1,A
     MOV
     MOV
             A,OP_2
     RLC
            A
OP_2,A
     MOV
            A, OP_3
     MOV
     RLC
            A
OP_3,A
     MOV
     RET
Shift_Q: ;shift the quotent one bit to the left and shift the C into LSB MOV A, TMP_0
     RLC
             TMP 0,A
     MOV
     MOV
             A,TMP_1
     RLC
             TMP_1,A
A,TMP_2
     MOV
     MOV
            A
TMP_2,A
A,TMP_3
     RLC
     MOV
     MOV
RLC
            A
TMP_3,A
     MOV
     RET
     END
                                                                                                  270530-7
```

int_d.

INTEL CORPORATION, 2200 Mission College Blvd., Santa Clara, CA 95052; Tel. (408) 765-8080 INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511